

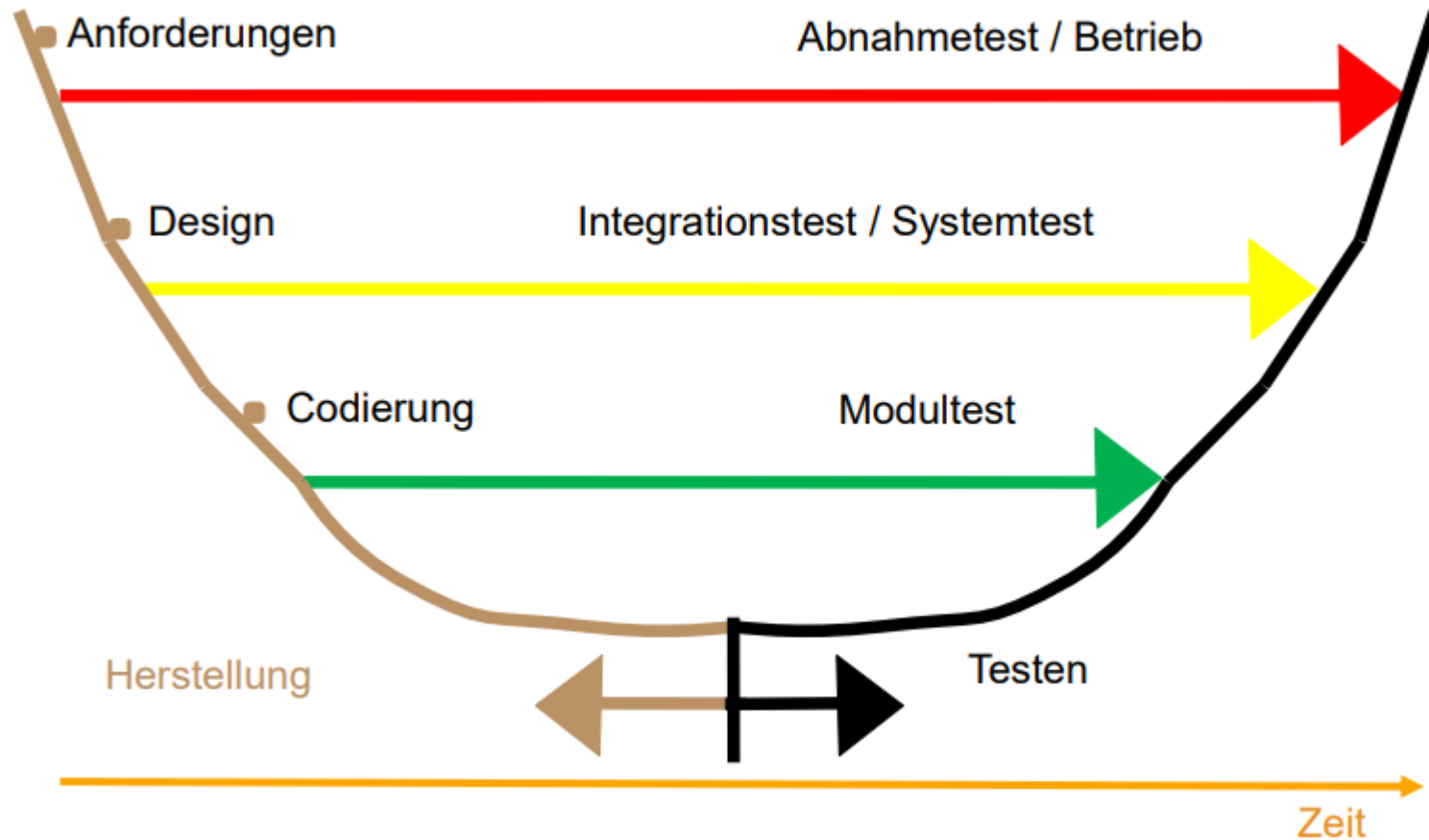
# Software Engineering – Testen

Systematisches Testen von Software

**HWZ**

Die Hochschule für Wirtschaft  
in Zürich

# Zusammenhang zwischen Fehler und Behebung



# Auf welcher Teststufe werden welche Fehler gefunden?

Teststufe	Codierungsfehler	Entwurfsfehler
<b>Modultest</b>	<b>65%</b>	0%
<b>Integrationstest &amp; Systemtest</b>	<b>30%</b>	<b>60%</b>
<b>Abnahmetest &amp; Betrieb</b>	3%	<b>35%</b>
<b>Total</b>	<b>98%</b>	<b>95%</b>

- Fehler werden auf der Entwicklungsstufe gefunden, auf der sie begangen wurden.

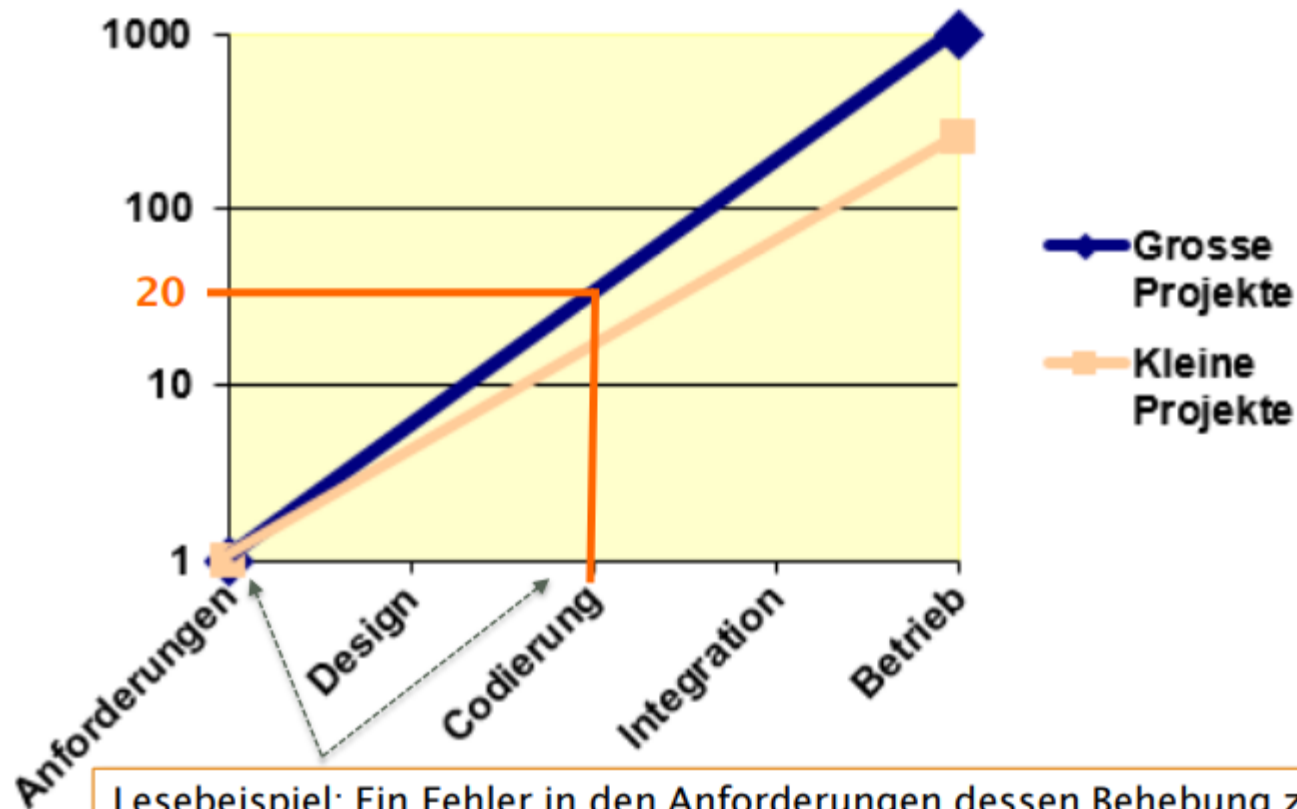
# Ursachen von Fehlerkosten verteilt auf die Projektphasen

- Analyse 46%
- Entwurf 27%
- Implementation 14%
- Wartung 13%

→ ca.  $\frac{3}{4}$  der Fehler finden der Analyse und Design – Phase statt.  
Und wie können sie solche Fehler vermieden / gefunden werden ?

# Relative Fehlerkostenentwicklung

- Die Kosten eines Fehlers in den Anforderungen steigen exponentiell zu seiner Verweildauer im Code



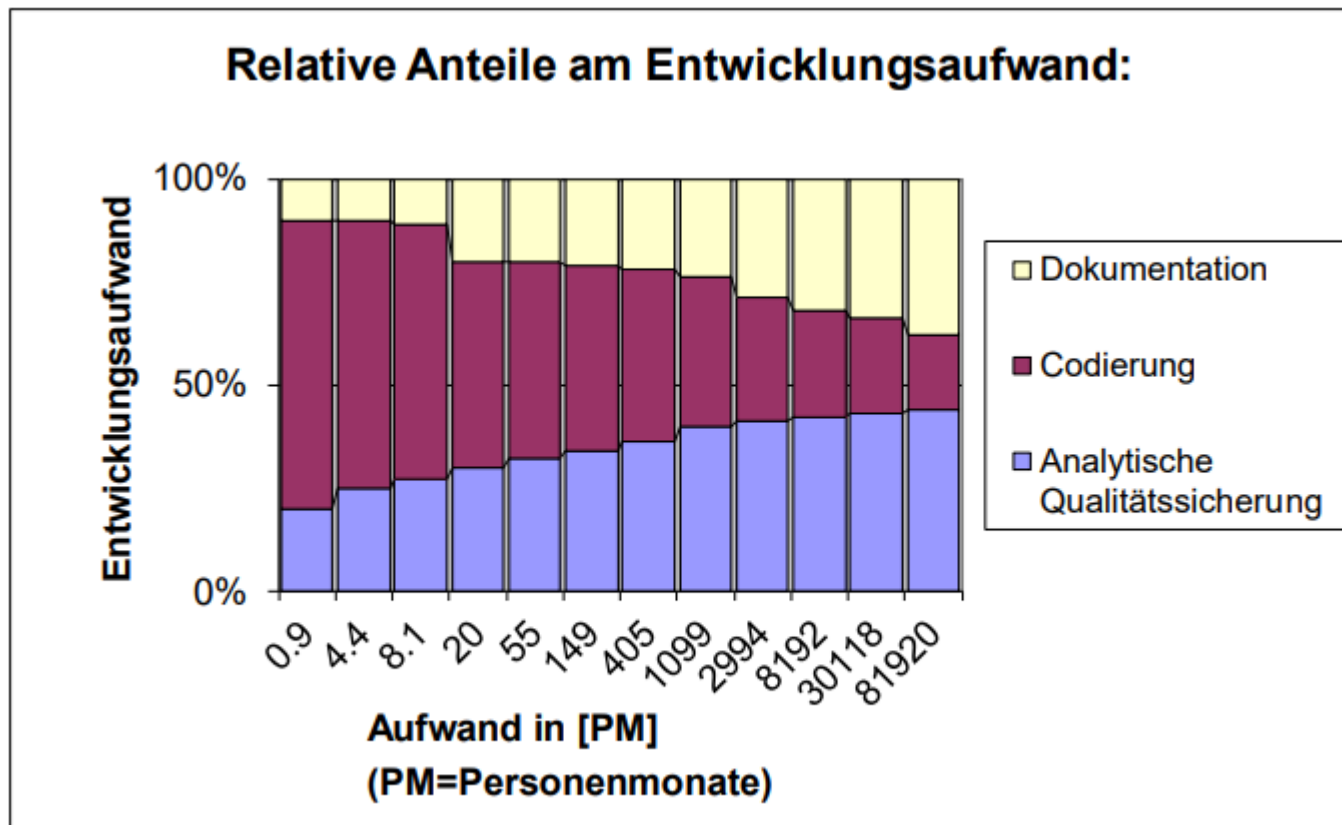
Lesebeispiel: Ein Fehler in den Anforderungen dessen Behebung zum Zeitpunkt des Anforderungsreviews 1 Fr. gekostet hätte, kostet zum Zeitpunkt der Codierung 20 Fr. (Bei einem grossen Projekt)

# Konsequenz zu Fehlerkosten

- Tests sind nicht die einzige Massnahme im Qualitätsmanagement der Softwareentwicklung, aber oft die letztmögliche (vor Inbetriebnahme)!
  - Je später Fehler entdeckt werden, desto aufwändiger ist ihre Behebung.
  - Daraus lässt sich der Umkehrschluss ableiten: Qualität muss im ganzen Projektverlauf implementiert werden und kann nicht am Schluss 'eingetestet' werden.
- Spezialfall Softwareentwicklung?
  - Beim Testen in der Softwareentwicklung wird in der Regel eine mehr oder minder grosse Fehleranzahl als 'normal' akzeptiert.
  - Hier herrscht ein erheblicher Unterschied zur Industrie: Dort werden im Prozessabschnitt 'Qualitätskontrolle' oft nur noch in Extremsituationen Fehler erwartet.

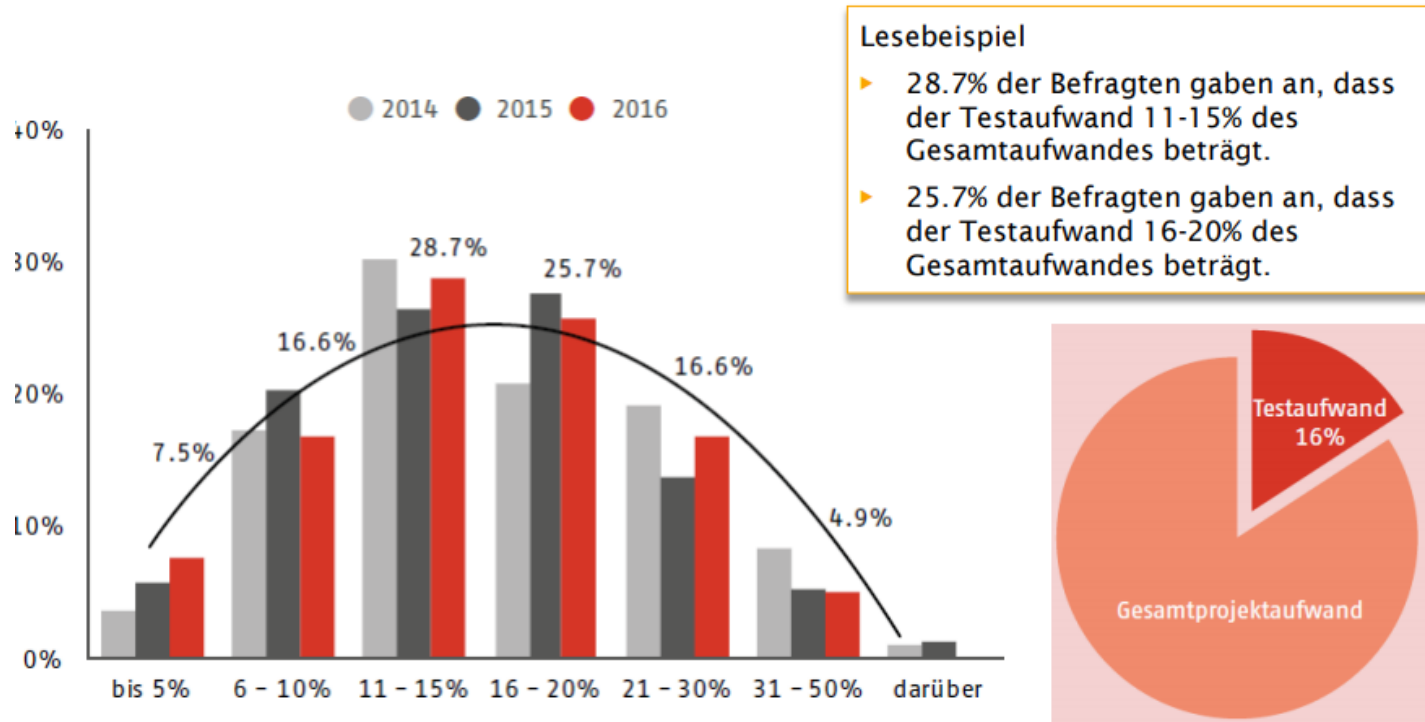
# Wie hoch ist der Anteil der QS-Massnahmen am Gesamtentwicklungsaufwand?

- Je grösser das Projekt, desto wichtiger wird Qualitätssicherung und Dokumentation



# Testaufwand im Verhältnis zum Gesamtaufwand

- Im Verhältnis zum Gesamtprojektaufwand wird der Testaufwand im Durchschnitt der Jahre auf ungefähr 16% geschätzt.
- Im Verhältnis zum Entwicklungsaufwand wird der Testaufwand im Durchschnitt der Jahre auf ungefähr 25% geschätzt.





# Verifikation

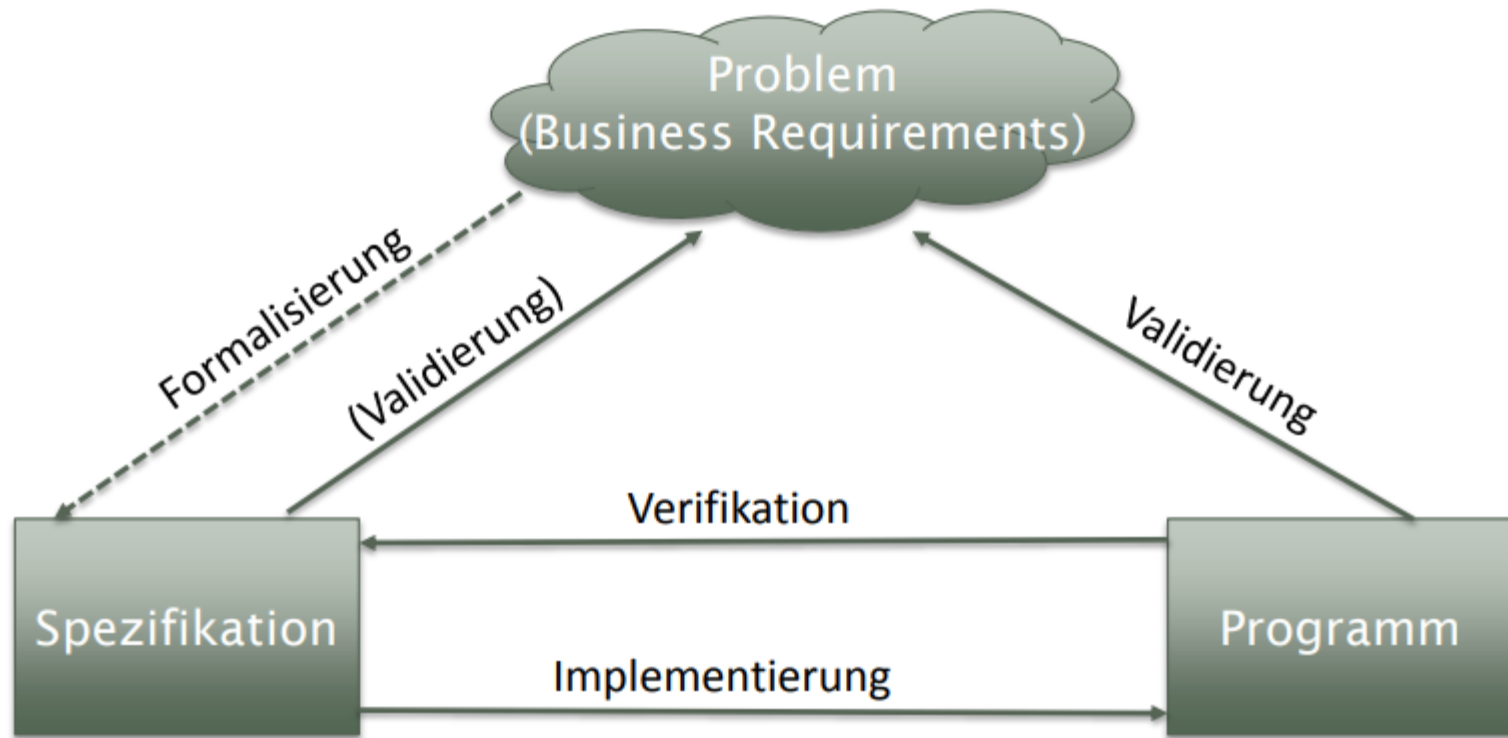
- Ziel
  - Überprüfen ob das Lieferergebnis einer Phase (Spezifikation, Komponente, System) funktional äquivalent zur vorherigen Phase ist.
  - Formale Prüfung ob die Lieferung der zugrundeliegenden Spezifikation entspricht.
- Problem
  - Kundenanforderungen sind meist nicht formal, deshalb sind sie kein geeigneter Input für die Verifikation.
  - Spezifikationen sind nur selten vollständig
  - Nur die nach der Spezifikationserstellung entstandenen Fehler können gefunden werden. Deshalb kein Nachweis der Fehlerfreiheit.
- Techniken zur Verifikation
  - Alle formalen Prüfungen (Audit, Review, formale Tests)

# Validation

- Ziel
  - Überprüfen ob das Lieferergebnis einer Phase (Spezifikation, Komponente, System) konsistent ist mit den Anforderungen des Kunden und diese erfüllt
- Problem
  - Anforderungen meist informell, unpräzise, unvollständig, widersprüchlich.
  - Automatisierung der Prüfung erst ab Stufe Implementation möglich.
- Techniken zur Verifikation
  - Review der Spezifikation.
  - How to demo erstellen (dadurch indirekt Validierung der Anforderungen).
  - Sprint Review (Validieren der Implementation).
  - Acceptance Tests (Abnahmetests durch den Kunden).
  - Systemtests soweit diese Funktionale / Nichtfunktionale Anforderungen testen.

# Validierung vs. Verifikation


- Nur die Validierung prüft gegen das zugrunde liegende Problem



```
graph TD; AD[Anforderungs-Definition] -- "(Validierung)" --> FS[Funktionaler Systementwurf]; FS -- "Statisch testbar (ohne Programmausführung)" --> TS[Technischer Systementwurf]; TS -- "Statisch testbar (ohne Programmausführung)" --> KS[Komponenten-Spezifikation]; KS --> P[Programmierung]; P --> AT[Abnahmetest]; P --> ST[Systemtest]; P --> IT[Integrationstest]; P --> KT[Komponententest]; AT -- "Validierung" --> AD; ST -- "Verifizierung" --> FS; IT -- "Verifizierung" --> TS; KT -- "Verifizierung" --> KS; AT --> ST; ST --> IT; IT --> KT; KT --> P; AT --> P; AT --> Exit(( ));
```

Das Diagramm stellt das V-Modell der Softwareentwicklung dar, das die Symmetrie zwischen Entwicklung und Test betont. Die linke Seite des 'V' zeigt die aufsteigende Entwicklung: von der **Anforderungs-Definition** über den **Funktionalen Systementwurf** und den **Technischen Systementwurf** zur **Komponenten-Spezifikation** und schließlich zur **Programmierung**. Die rechte Seite zeigt die absteigenden Testaktivitäten: **Abnahmetest**, **Systemtest**, **Integrationstest**, **Komponententest** und schließlich die Rückführung zur **Programmierung**. Horizontale gestrichelte Pfeile mit den Beschriftungen **Validierung** (zwischen Anforderungs-Definition und Abnahmetest) und **Verifizierung** (zwischen den entsprechenden Entwurfsebenen und Testaktivitäten) verdeutlichen die Verknüpfung. Ein großer grauer Pfeil am oberen Rand zeigt nach rechts, was den Fortschritt der Entwicklung symbolisiert. Zusätzliche diagonale Beschriftungen auf der linken Seite (**Statisch testbar (ohne Programmausführung)**) und rechten Seite (**Dynamisch testbar (mit Programmausführung)**) unterscheiden die Testarten. Ein kleiner grauer Pfeil am unteren Rand zeigt nach rechts, was den Abschluss des Prozesses andeutet.

# Beschreibung der Teststufen




Der **Abnahmetest** prüft...  
...ob das System aus Kundensicht die vertraglich vereinbarten Leistungsmerkmale aufweist

Der **Systemtest** prüft...  
...ob das System als Ganzes die spezifizierten funktionalen und nichtfunktionale Anforderungen erfüllt.

Der **Integrationstest** prüft...  
...ob Gruppen von Komponenten gemäss technischem Systementwurf zusammenspielen

Der **Komponententest** prüft...  
...ob jede einzelne Komponente für sich die Vorgaben seiner Spezifikation erfüllt



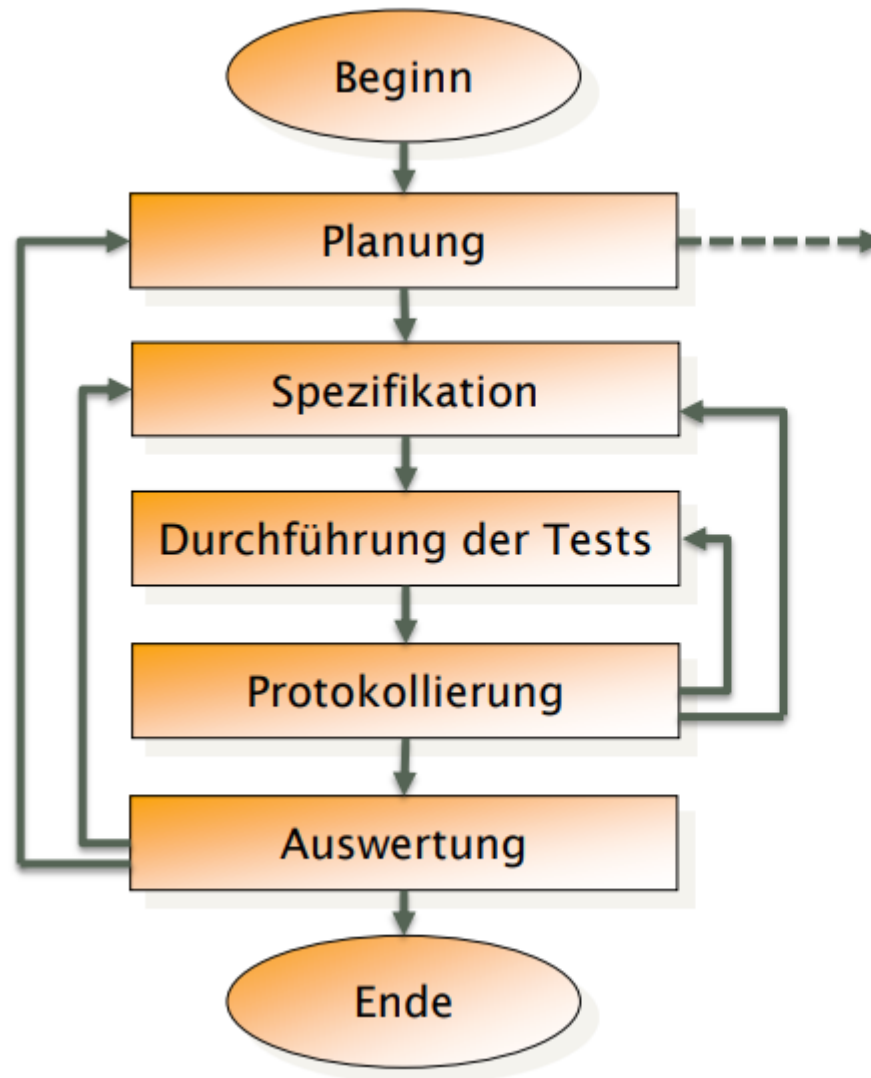
# Testarten

- Informelle und formelle Test
- Big Bag vs. inkrementelles Testen
- Top Down v. Bottom Up Testen
- WhiteBox vs BlackBox Testen

Übung: Testarten – 2er Gruppen je Begriff:  
15 min. Zeit, 3 min. Präsentation

- Testen nichtfunktionaler Anforderungen
  - Lasttest, Volumentest, Massentest,
  - Performancetest, Stresstest und weitere

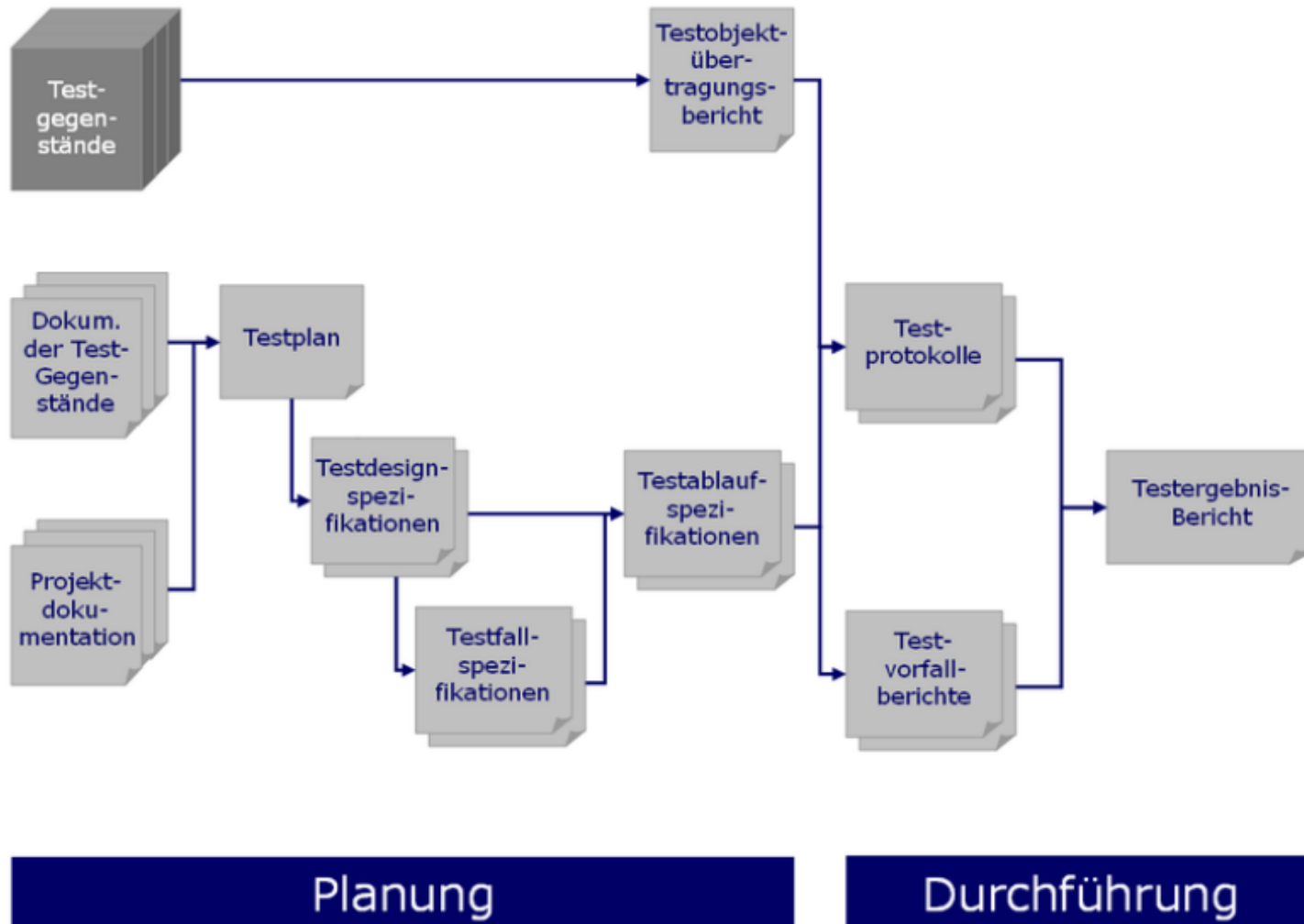
# Fundamentaler Testprozess



## Erstellen der Teststrategie:

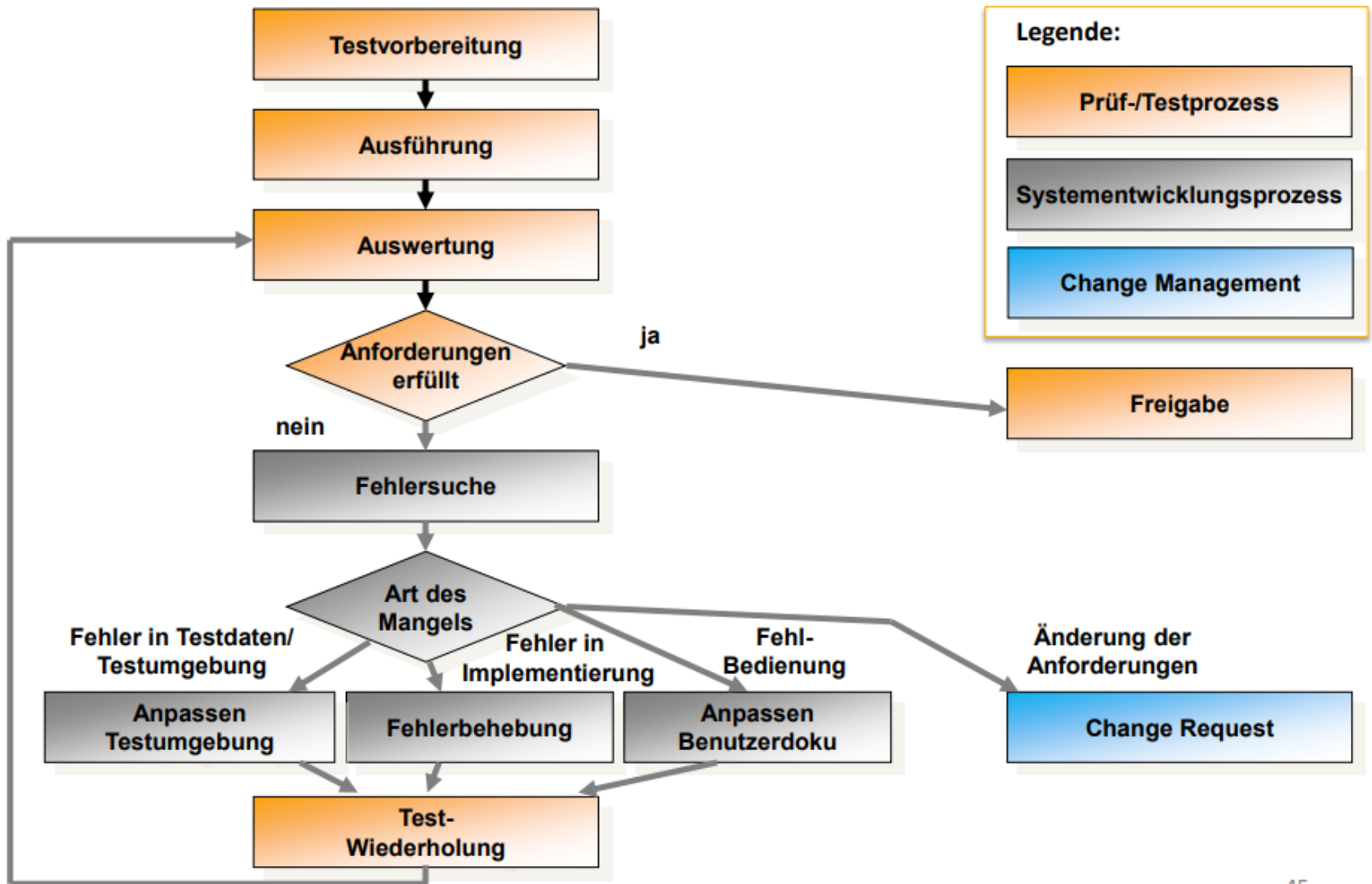
- Mittels Risikoabschätzung beurteilen, wie kritisch das Auftreten eines Fehlers in einem Systemteil ist.
- Unter Berücksichtigung der verfügbaren Ressourcen definieren, wie intensiv jedes Systemteil getestet werden kann oder muss.
- Definieren mit welchem Vorgehen und mit welcher Priorität jedes Systemteil getestet werden soll.

# Zusammenhang der Dokumente und Verfahrensschritte nach IEEE 829

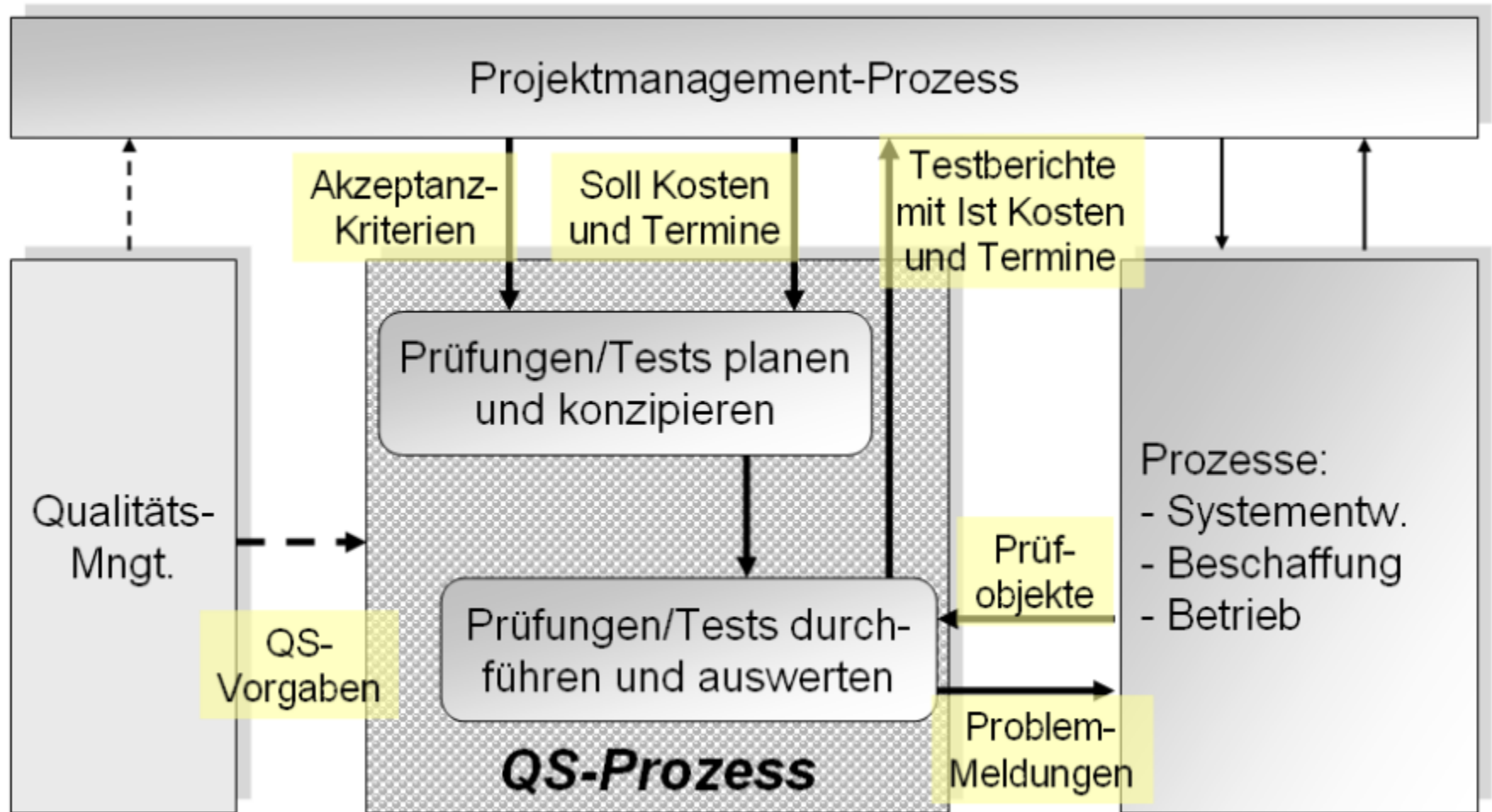




# Schnittstellen zum Softwareentwicklungsprozess



# Schnittstellen zum Projektmanagement Prozess



# Erfolgsfaktoren Testing

- Die frühe Involvierung des Testings wird als wichtigster Erfolgsfaktor angesehen.
- Wenn das Testing bereits bei der Anforderungserhebung aktiv beteiligt ist, steigt die Qualität der Anforderungen und das Know-How, welche wiederum zu den wichtigsten Erfolgsfaktoren gezählt werden.
- Methodisches Vorgehen
- Stabile Umgebung
- Verfügbarkeit von Testdaten